

Evidence Management in Programatica

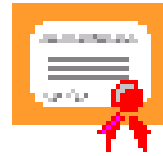
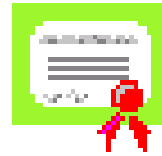
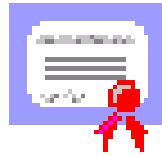
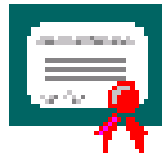
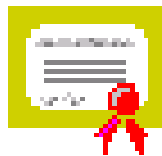
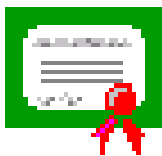
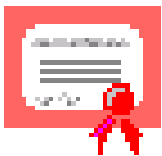
(Presentation for SoftCeMent '05)

Mark P Jones

Portland State University

November 2005

(joint work with the Programatica Project at PSU and OGI/OHSU)



Programmatica Positions:

- **“Programming as if Properties Matter”** to support the construction and certification of high-assurance systems
- **There is a Broad Spectrum of (Useful) Assurance Techniques:** code review, testing, formal methods, ...
- **Everything Changes:** flexible and efficient tools are needed to deal with constantly evolving requirements, code, evidence, and assurance goals
- **Make it Real:** assuring security properties of a real-world microkernel implementation

Programming as if Properties Matter:

The screenshot shows the Programatica Haskell Browser interface. On the left is a 'Module Graph' sidebar with a tree of modules: Files (Bungalow.Ihs, Decoder.Ihs, HW.Ihs, IOMonad.Ihs, Machine.Ihs, MonadT.Ihs, Perms.Ihs, PhysMem.Ihs, StateMonad.I, Utils.Ihs, VirtMem.Ihs) and Modules (hi). The main window displays the source code for 'HW.Ihs'. The code defines a 'preserveMode' function and includes a comment explaining its purpose: 'Instead of exporting an operation that switches between user and kernel mode, we provide only the following \hs{preserveMode} which executes a command \hs{c} and then restores the previous mode.' Below the code, there are two sections: 'Properties' and 'Certificates'. The 'Properties' section contains two assertions: 'assert PreserveModeIdempotent' and 'assert PreserveModeDist'. The 'Certificates' section shows the type signature for 'PreserveModeIdempotent' and a certificate for the 'forall' property.

File: HW.Ihs

Module: HW

Source Code

```
Instead of exporting an operation that switches between user and kernel mode, we provide only the following \hs{preserveMode} which executes a command \hs{c} and then restores the previous mode.
```

```
> preserveMode :: HW m a -> HW m a
> preserveMode c = do md <- getMode
>                   x <- c
>                   setMode md
>                   return x
```

A program that is already running in kernel mode can use a command of the form \hs{preserveMode (userMode >> c)} to execute \hs{c} in user mode and then switch back to kernel mode. However, there is no way for a program that is running in user mode to switch to kernel mode using any combination of \hs{userMode} and \hs{preserveMode}.

%% Nested uses of \hs{preserveMode} behave the same as

```
! assert PreserveModeIdempotent
! = {preserveMode . preserveMode} == preserveMode

! assert PreserveModeDist
! = All c, f. {preserveMode (c >> f)}
! ==
! = f . preserveMode
```

Properties

```
PreserveModeIdempotent: HW.PreserveModeIdempotent, Assertion
Certificates: none. Create a new certificate!
forall {(a::*->*) b}. Prelude.Prop
```

Building High-assurance Software:

There are many ways to increase assurance:

- Test programs on specific cases
- Test programs on randomly generated test cases derived from expected properties
- Peer review
- Use algorithms from published papers
- Reason about meta-properties (e.g., using types)
- Use theorem provers to validate (translated) code
- ...

Each can contribute significantly to increased reliability, security, and trustworthiness

Evidence:

- Diverse techniques, varying in:
 - Applicability
 - Assurance
 - Technical details
- But there is a common feature:
 - Each one results in some **tangible** form of **evidence** that provides a **basis for trust**

Examples of Evidence:

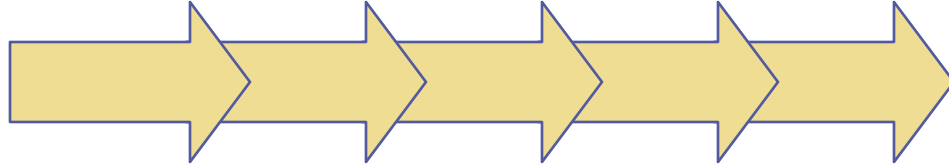
There are many kinds of evidence:

- An (input, expected output) pair for a test case
- A property statement, and heuristics for guiding the selection of “interesting” random test cases
- A record of a code review meeting
- A citation/URL for a published paper or result
- A type and the associated derived property
- A translation of the source program into a suitable theory and a user-specified proof tactic
- ...

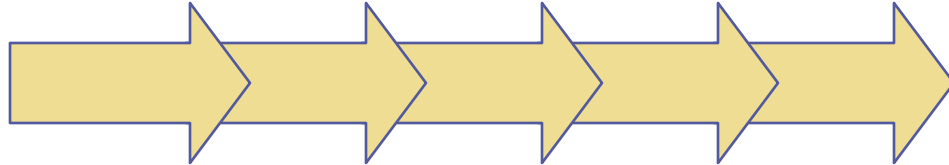
Each different kind of evidence is stored with the program as a **certificate**

Extreme Programming:

Tests

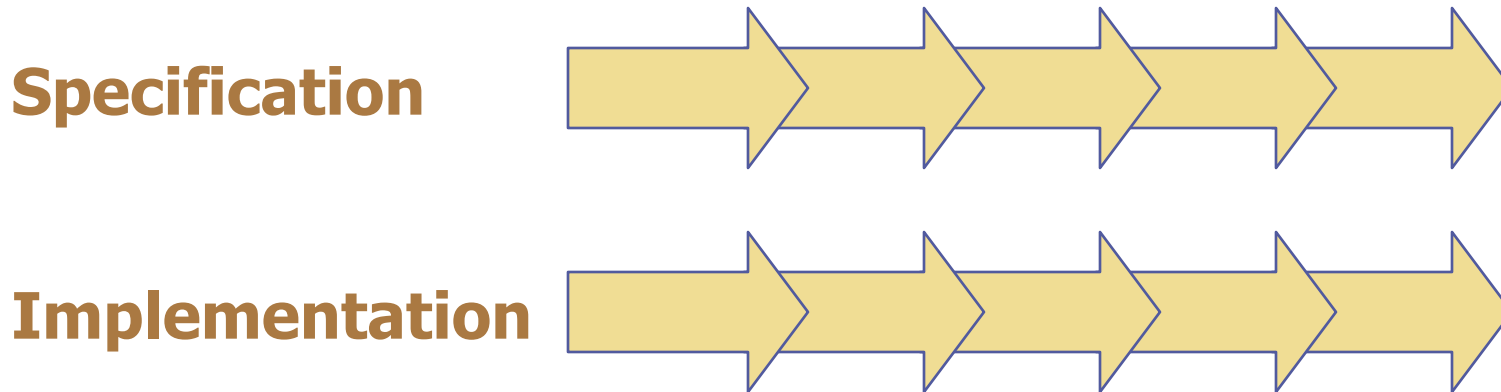


Implementation



- Testing and Programming, hand in hand
- Testing reveals errors in the program
- Programming reveals errors in the test cases

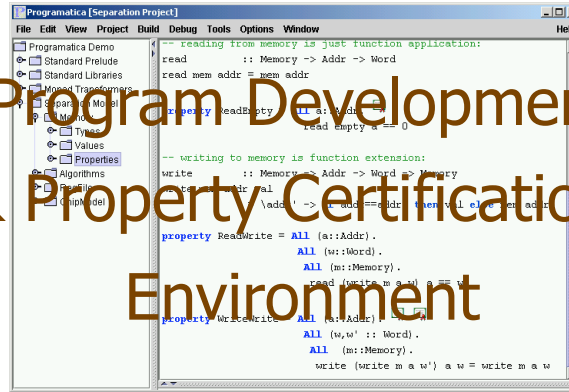
"Extreme Formal Methods":



- Programming and Validation, hand in hand
- Validation reveals errors in the program
- Programming reveals errors in the specification

The Programatica Vision:

Program Development & Property Certification Environment



Type checking

Execute & test

Random test generator

Code review

Instrumenting compiler

Automatic Decision Procedures

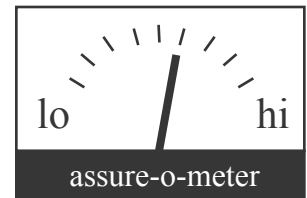
Model Checking

Reporting, Analysis, Management







User supplied, domain-specific toolsets...

Interactive Proof Editor

Theorem Proving



Programatica Servers:

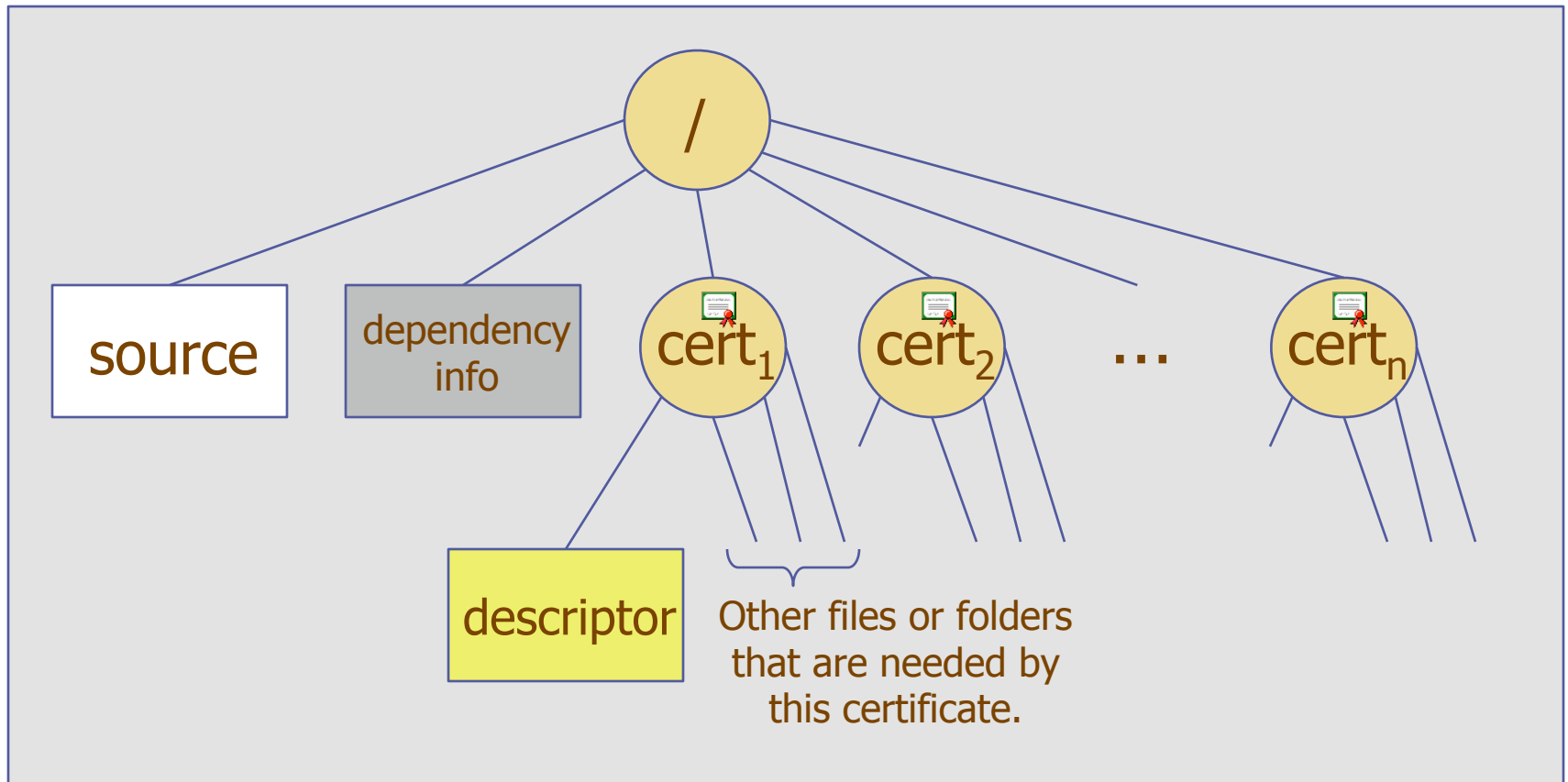
- “I say so” 
 - A person signs their name by an assertion
 - Testcases 
 - Individual test cases / regression testing
 - QuickCheck 
 - Random testing
 - Plover 
 - The P-logic verifier
 - Alfa 
 - Interactive proof editor based on type theory
 - Isabelle 
 - Logical framework, tactic-based theorem prover
- implemented,
automated,
maturing
- hand / auto
translation

Evidence and Certificates:

The certificate abstraction is designed to support:

- **Capturing** evidence (in many different forms) and **Collating** it with source materials
- **Combining** evidence from different sources
- **Tracking** dependencies and **detecting** when evidence must be revalidated as a result of changes
- **Managing** evidence by analyzing and reporting on what has been established, identifying weaknesses, guiding further effort, etc...

Capture and Collate:



Compound documents allow source materials to be packaged with related evidence and dependency information.

Combining Evidence:

Programatica allows us to combine evidence from different sources:

Goals:

- Evidence Integration
- Modular Certification

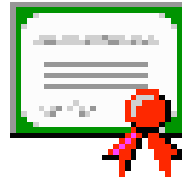
Mechanism:

- Each certificate carries a **sequent**:
$$\text{Hypotheses} \vdash \text{Conclusions}$$
- Servers for external tools are responsible for testing **validity** (i.e., checking that a certificate's sequent is consistent with its evidence)

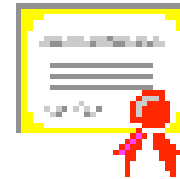
Certificate Interactions:

N.B. Different kinds of certificate

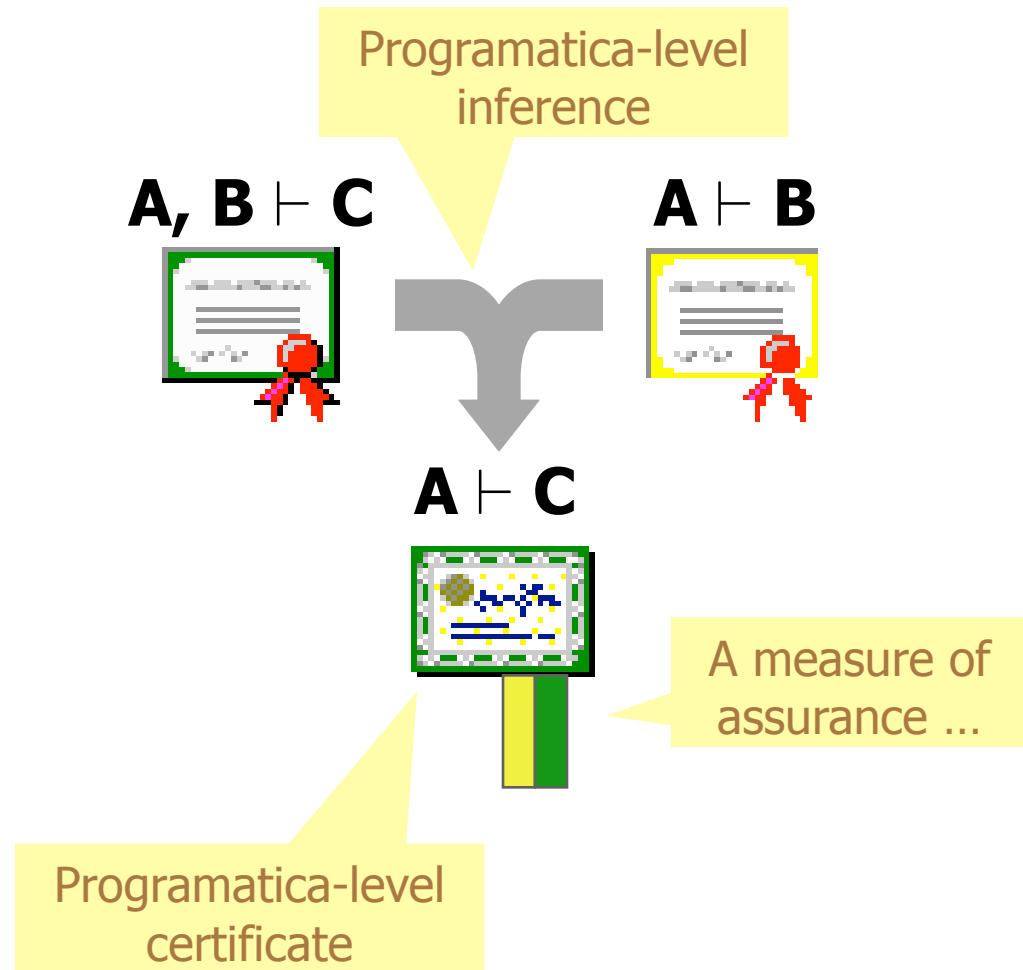
$A, B \vdash C$



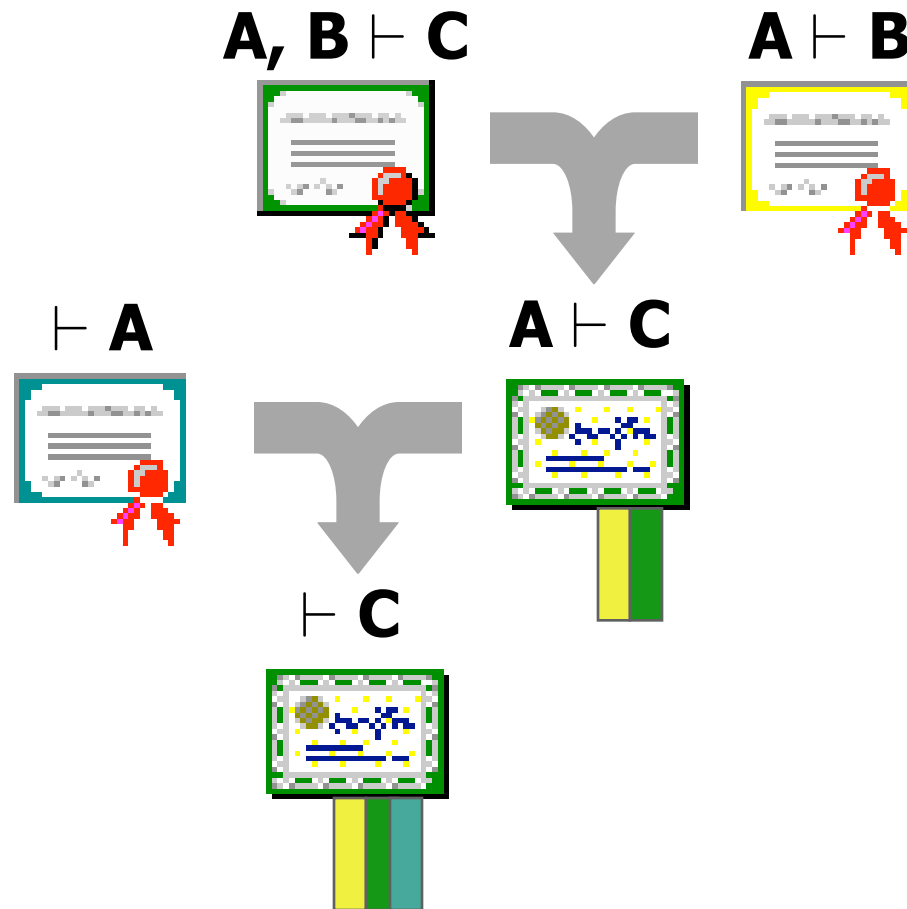
$A \vdash B$



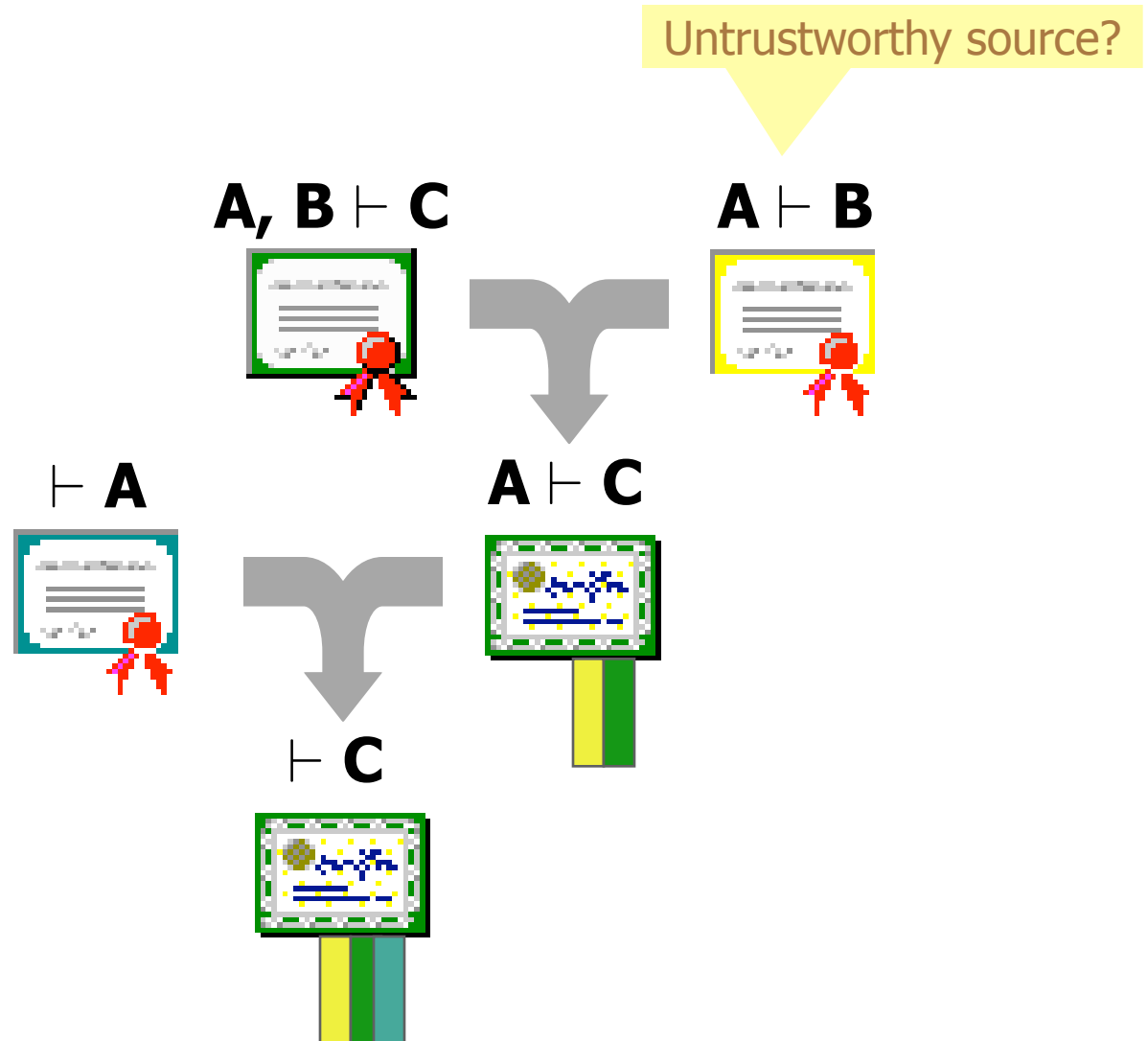
Certificate Interactions:



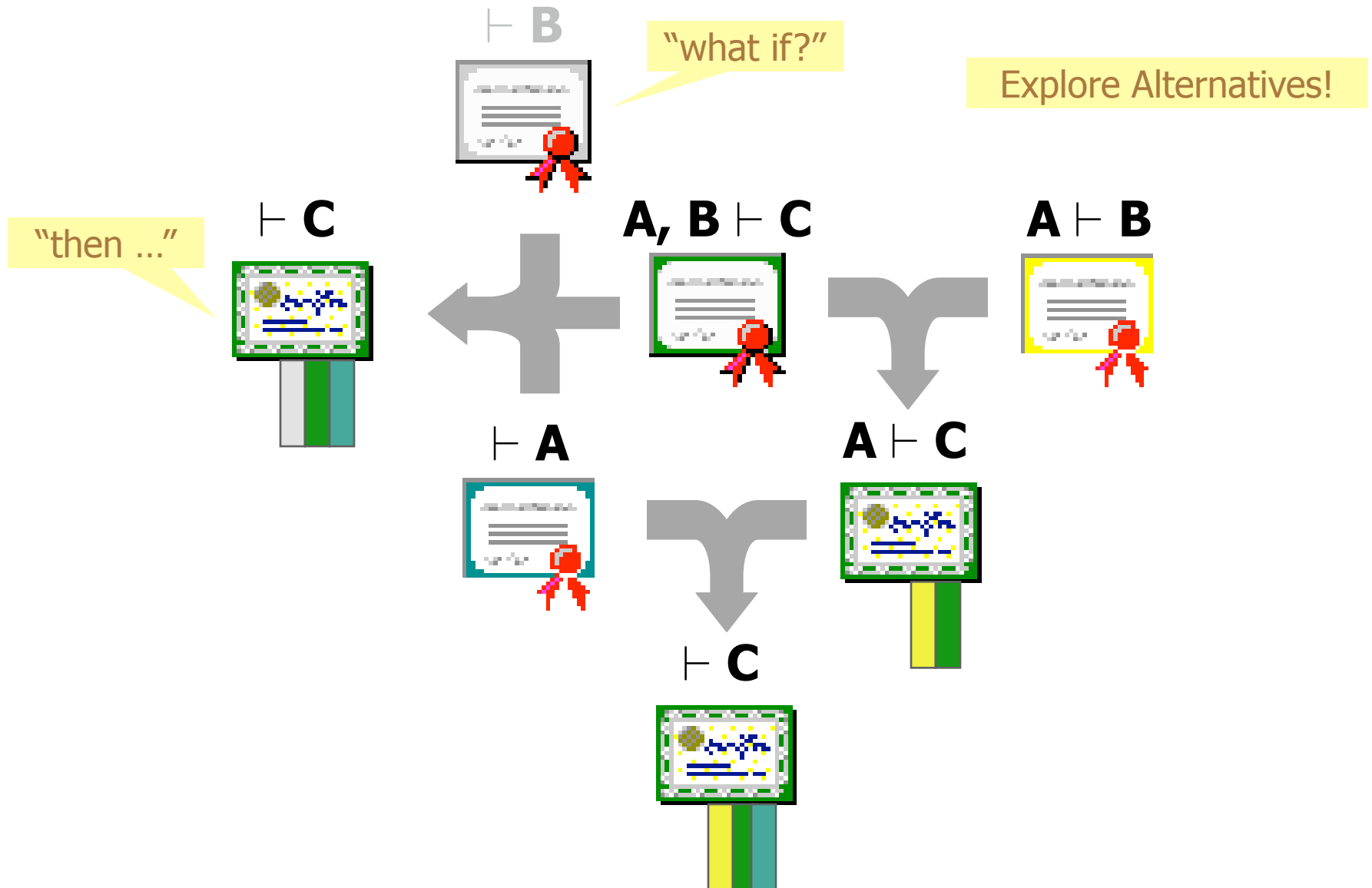
Certificate Interactions:



Certificate Interactions:



Certificate Interactions:



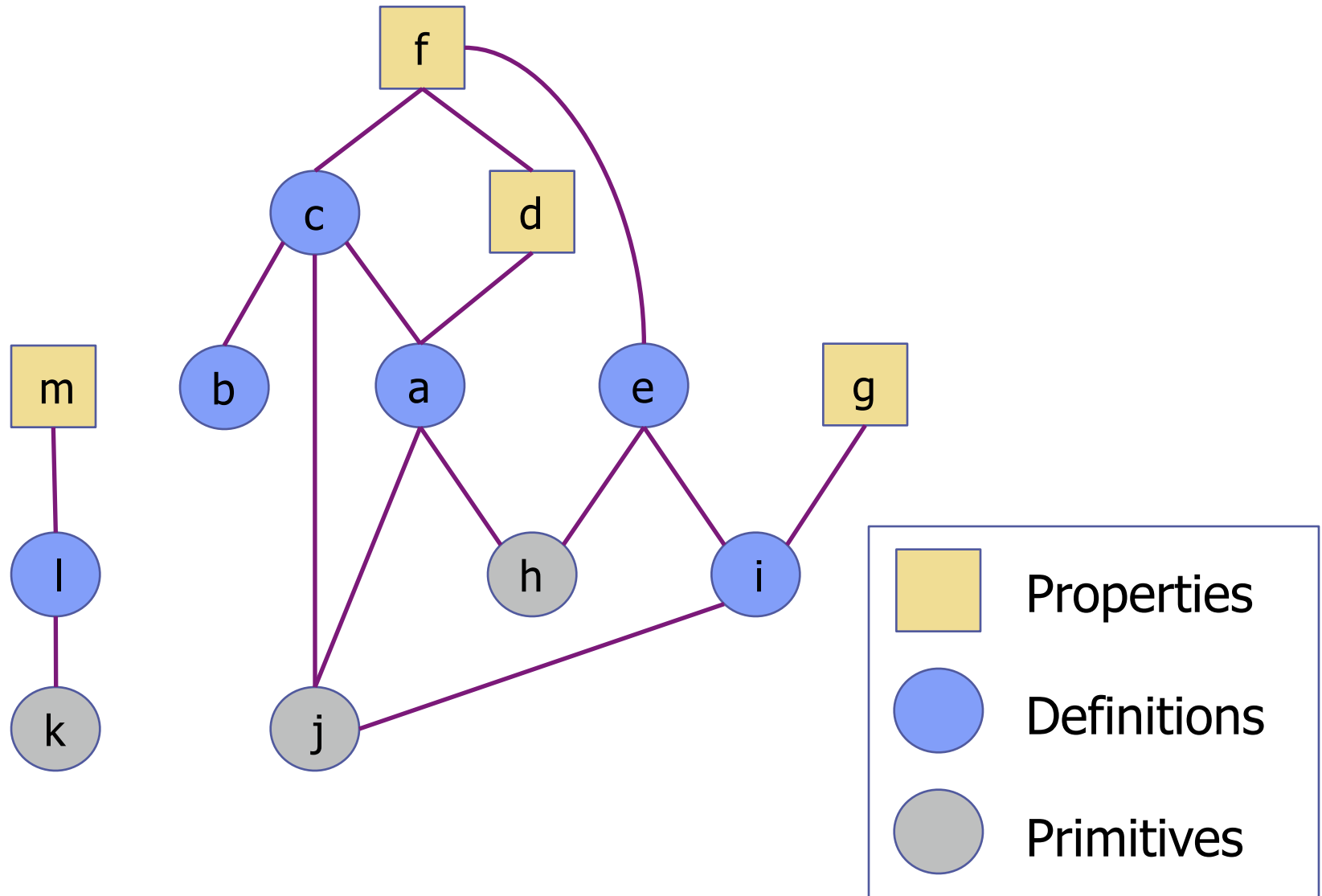
Dealing with Change:

- Changes happen all the time in software development!
 - functionality, requirements, bug fixes, assurance
- We must handle change as efficiently as possible
- Changes to source code require recompilation
 - A fully automated process using “make” tools
- Changes to source code require recertification
 - Some evidence cannot be reconstructed automatically

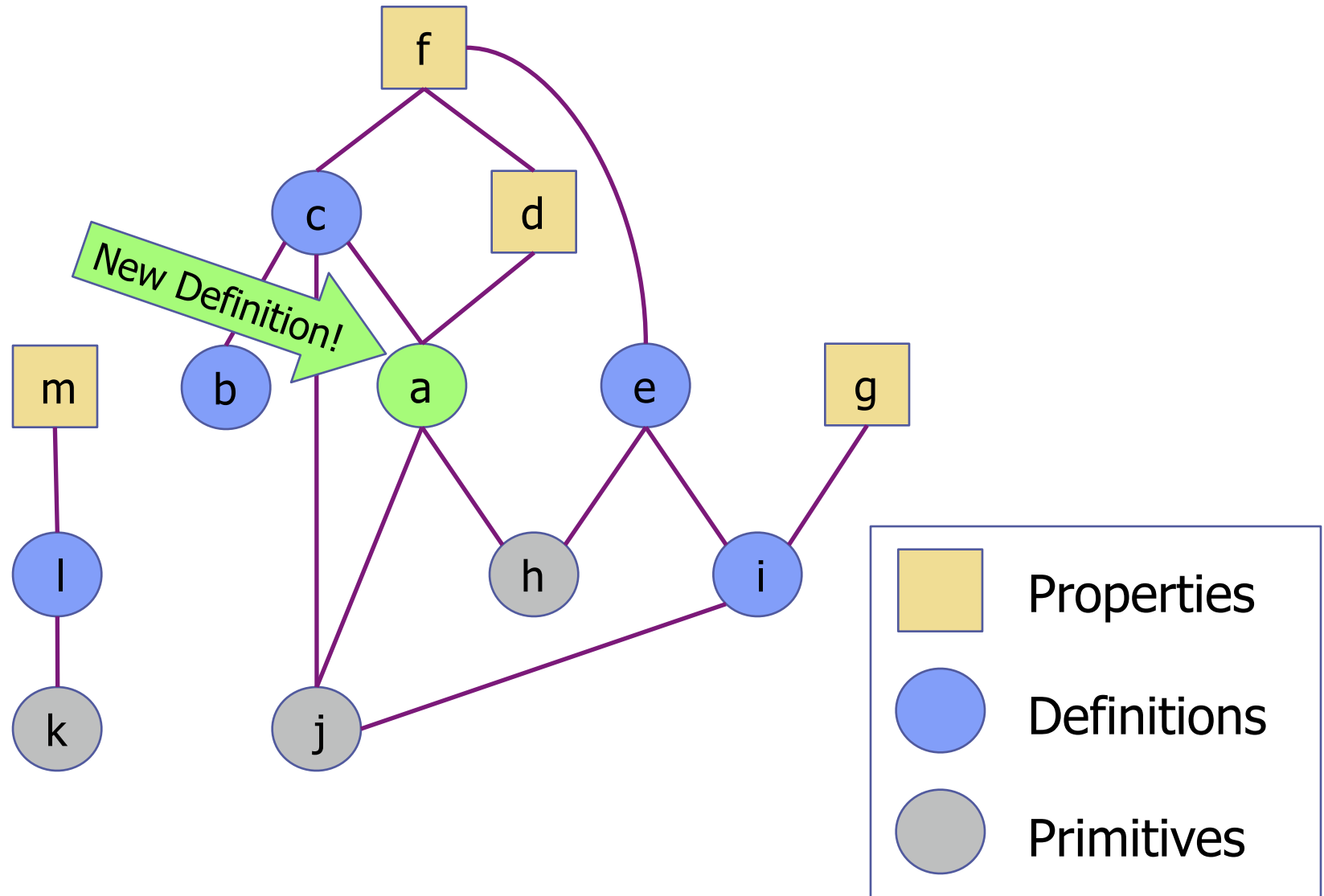
Recertification after Change:

- “make”-like functionality for certification
 - Track dependencies to determine when evidence is invalidated by changes to source code
- Minimize the need for recertification:
 - Fine-grained dependency tracking
 - Robust dependency tracking
 - Ignore insignificant changes: reformatting; reordering; changes to comments; changes to local variable names; changes in unrelated sections of code; ...
 - Lazy recertification
 - Track validity but do not require immediate recertification

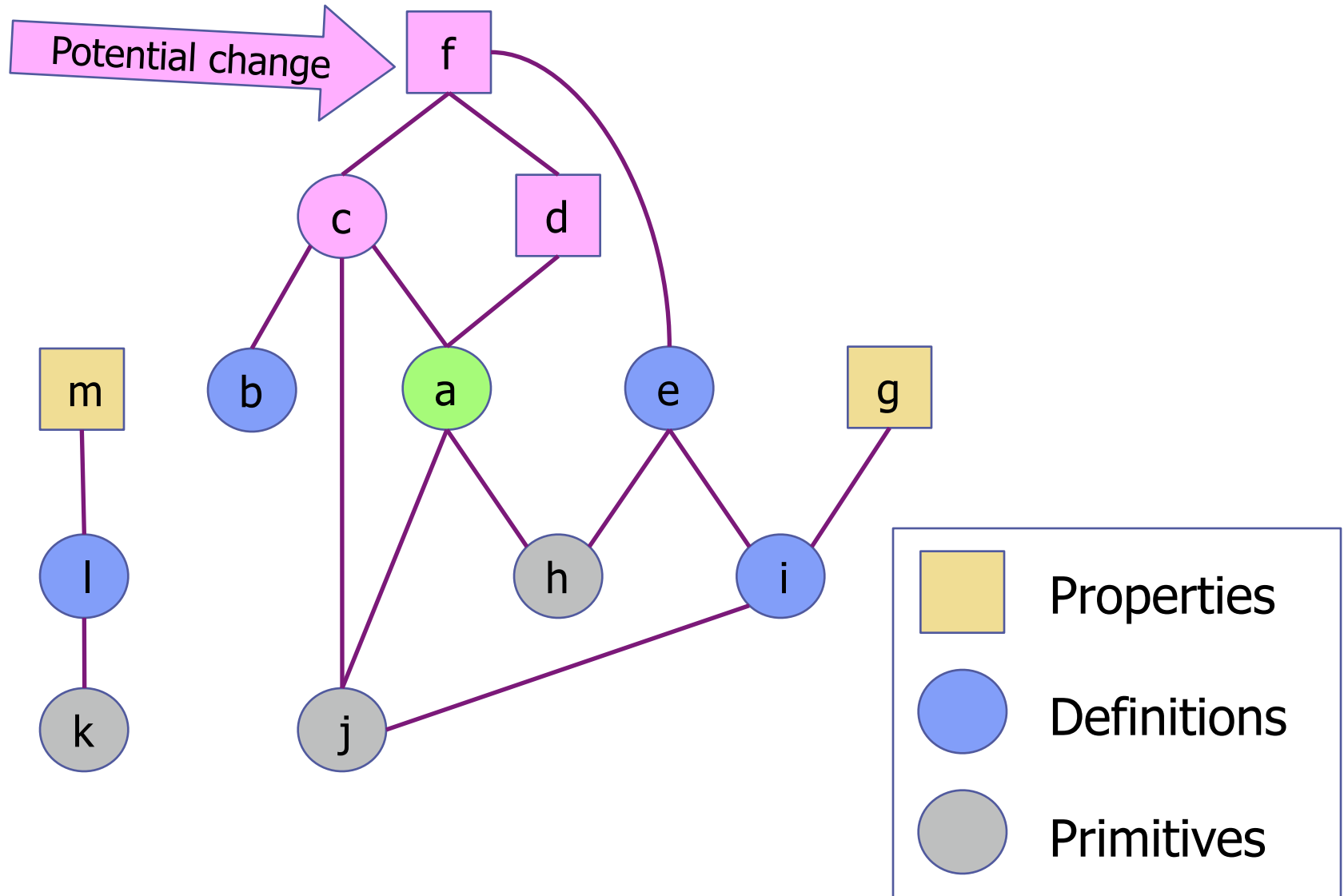
Using a Dependency Graph:



Using a Dependency Graph:



Using a Dependency Graph:



Hashing to Detect Change:

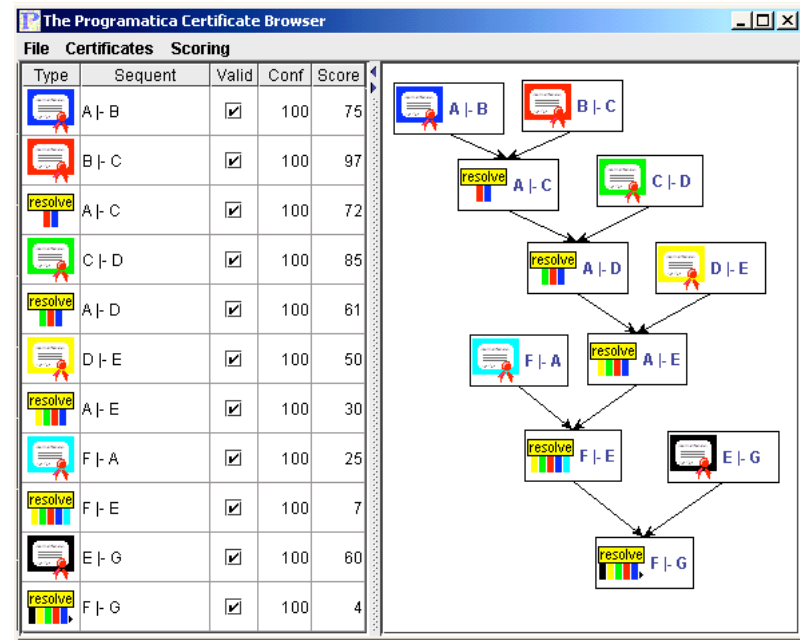
- When we parse a source file, we calculate a cryptographically robust hash (e.g., MD5) over the abstract syntax of each definition
- These hashes are cached as hidden information:

```
0cc175b9c0f1b6a831c399e269772661  
92eb5ffee6ae2fec3ad71c777531578f  
81a5fe3d544359af13848e6192ece475  
445a4ca24e10824e03ef42e2e1d755d9  
987dd8f5f1293857dc7932c14c7f3d80  
bb53046df3ef7793ee7c37aec0d090d0  
ad797e6f29cf558f7aeb8200563ecd3a
```
- If we find a definition whose hash is not listed, then it must be new/modified.
- By hashing over abstract syntax, we do not flag any changes if the source text is reformatted, if comments are changed, etc...

Management Tools:

Certificate management tools let users ask (and answer) questions like the following:

- What properties have I verified (or not)?
- What tools did I use?
- Is the evidence up to date & consistent with the code?
- What conclusions can we draw from the evidence in hand?
- What other verification strategies should I pursue?
- Where am I most vulnerable?
- What should I do next?



Scoring & prioritization mechanisms required

Future Challenges:

- Making the assure-o-meter real
- Dealing with non-functional properties
- Encoding certification policy
- Certifying the certification tools ...
- Developer Carrot and Stick